МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: Изучение деревьев

Вариант 3

Студент гр. 8304 ___ Птухов Д. А.

Преподаватель ___ Фиалковский М. С.

Санкт-Петербург

2019

Цель работы.

Получить опыт работы с деревьями (реализация на основе списка).

Постановка задачи.

Для заданного бинарного дерева с произвольным типом элементов, определить, есть ли в дереве хотя бы два одинаковых элемента.

Описание алгоритма.

1) Считывание осуществляется при помощи конструкции:

```
while (std::getline(in, stringTreeForm))
```

2) После считывания очередной строки из переданного файла, при помощи функции formTree создается бинарное дерево, голова которого заносится в переменную root.

3) При помощи 3-х рекурсивных функций, взаимодействующих между собой, осуществляется поиск двух одинаковых элементов в дереве.

Спецификация программы.

Программа предназначена для нахождения хотя бы двух одинаковых элементов в заданном дереве.

Программа написана на языке C++.

Описание функций и структур данных.

1) Для хранения узлов дерева была реализована структура Node.

   Данная структура содержит 3 поля – value, left, right. Value – значение хранящееся в данном узле, left и right – правое и левое поддерево соответственно.

2) Для составления данных из полученной строки используется рекурсивная функция formTree, принимающая рассматриваемый узел и

подстроку исходной строки, содержащую данные, которые необходимо внести в рассматриваемый узел. Заполнение реализовано при помощи нескольких циклов while и рекурсивного вызова функции formTree для левого и правого поддерева.

3) Для решения поставленной подзадачи были реализованы 3 рекурсивные функции – checkElementsOnSameness, treeSearch, findSameElements. Первой запускается последняя функция в списке, она рекурсивно перебирает все узлы дерева и для каждого не пустого узла вызывает функцию treeSearch. Функция treeSearch также рекурсивно проходится по дереву однако хранит в себе искомый в данном дереве элемент. При помощи функции checkElementsOnSameness она определяет равен ли текущий рассматриваемый элемент искомому. Функция checkElementsOnSameness проверяет переданные ей элементы на равенство их значений и рекурсивно делает ту же операцию с левыми и правыми поддеревьями рассматриваемых элементов.

Тестирование.

Таблица 1 – Результаты тестирования программы

Input	Output
(a(b(c(t)(y)))(e(a(c(u)(teee)))))	NO
(a(b(a)(a2))(c(a1)(a)))	YES
(a(c)(c))	YES
()	Hm, your tree is incorrect or so hard for me (i can read only binaries trees)
(aaaaaaaa (bbbb (cccc)) (cccc))	YES
(Please(Say(YES))(Okay(I(Will(Say(YES))))))	YES
)()(()())()(	Incorrect brackets placement!
/\	Incorrect brackets placement!
{a{b}{}}	Hm, your tree is incorrect or so hard for me (i can

	read only binaries trees)
(a()(b))	Hm, your tree is incorrect or so hard for me (i can read only binaries trees)

Выводы.

В ходе работы был получен опыт работы с деревом на основе списка.

Приложение А. Исходный код программы.

BinaryCheck.cpp

```
#include "BinaryTree.h"

bool checkTreeBrackets(std::string const& stringTreeForm)
{
	//ocnt - open cnt, ccnt - close cnt
	int ocnt = 0, ccnt = 0;

	for (char i : stringTreeForm)
	{
		if (i == '(')
			ocnt++;

		if (i == ')')
			ccnt++;

		//можно любое число > 0
		if (ccnt > ocnt)
			return false;
	}

	if (ocnt == 0 || ccnt == 0)
		return false;

	return ocnt == ccnt;
}

std::string extractBracketsValue(std::string const& stringTreeForm, size_t*
stringIndexPointer)
{
	//error - переменная необходимая для обработки данного случая (...(...)...) она
позволяет получить значение
	//лежащее точно от уже найденной открывающей до корректной закрывающей скобки
	int tmp_ind = *stringIndexPointer, error = 0;
	std::string bracketsValue;

	while (tmp_ind < stringTreeForm.size())
	{
		//запись очередного символа
		bracketsValue += stringTreeForm[tmp_ind];
		tmp_ind++;

		if (stringTreeForm[tmp_ind] == '(')
			error++;
		if (stringTreeForm[tmp_ind] == ')')
			error--;
		if (error < 0)
			break;
	}

	//запись ')'
	bracketsValue += stringTreeForm[tmp_ind];

	//перенос индекса за выражение в скобках дял считывания второго аргумента
	*stringIndexPointer = tmp_ind + 1;
	return bracketsValue;
}

bool formTree(std::string const& stringTreeForm, std::shared_ptr<Node>& root)
{
	//первый символ - '(', нет необходимости его рассматривать
	size_t stringIndex = 1;

	//запись имени корня
	std::string rootName;
	while (stringIndex < stringTreeForm.size() &&
		(stringTreeForm[stringIndex] != '(' && stringTreeForm[stringIndex] !=
')'))
	{
```

```
			rootName += stringTreeForm[stringIndex];
			stringIndex++;
		}
		if (rootName.empty())
			return 0;

		root->name = rootName;
		//

		//если был встречен конец строки, то левое и правое поддерево пустые
		if (stringTreeForm[stringIndex] == ')')
		{
			root->left = nullptr;
			root->right = nullptr;
			return true;
		}
		//

		auto leftTree = std::make_shared<Node>();
		std::string bracketsValue = extractBracketsValue(stringTreeForm, &stringIndex);

		bool formLeftResult = formTree(bracketsValue, leftTree);
		if (!formLeftResult)
			return false;

		//если был встречен конец строки, то правое поддерево пустое
		if (stringTreeForm[stringIndex] == ')')
		{
			root->left = leftTree;
			root->right = nullptr;
			return true;
		}
		//

		auto rightTree = std::make_shared<Node>();
		bracketsValue = extractBracketsValue(stringTreeForm, &stringIndex);

		bool formRightResult = formTree(bracketsValue, rightTree);
		if (!formRightResult)
			return false;

		//проверка на корректность конечных символов
		if (stringTreeForm[stringIndex] != ')' || stringIndex + 1 !=
stringTreeForm.size())
			return false;
		//

		//формирование бинарного дерева
		root->left = leftTree;
		root->right = rightTree;
		//

		return true;
}

void printTree(std::shared_ptr<Node> const& root, std::string const& mainRootName =
"")
{
		if (!root)
			return;

		if (root->left)
			std::cout << root->name + " : " + root->left->name;
		else
			std::cout << root->name + " - " + "leaf";

		if (root->right)
			std::cout << ", " + root->right->name + "\n";
		else
			std::cout << "\n";

		if (root->name == mainRootName)
			std::cout << "\nHey this a left part:\n";
```

```
	printTree(root->left);

	if (root->name == mainRootName)
		std::cout << "\nHey this a right part:\n";
	printTree(root->right);
}



bool findSameElements(std::shared_ptr<Node> const& checkElement,
std::shared_ptr<Node> const& mainRoot)
{
	if (checkElement == nullptr)
		return false;

	if (treeSearch(checkElement, mainRoot))
		return true;

	if (findSameElements(checkElement->left, mainRoot))
		return true;

	return findSameElements(checkElement->right, mainRoot);
}

bool treeSearch(std::shared_ptr<Node> const& checkElement, std::shared_ptr<Node>
const& root)
{
	if (!root)
		return false;

	if (&checkElement == &(root->left))
		return false;

	if (checkElementsOnSameness(checkElement, root->left))
		return true;

	if (&checkElement == &(root->right))
		return false;

	if (checkElementsOnSameness(checkElement, root->right))
		return true;

	if (treeSearch(checkElement, root->left))
		return true;

	return treeSearch(checkElement, root->right);
}

bool checkElementsOnSameness(std::shared_ptr<Node> const& firstElement,
std::shared_ptr<Node> const& secondElement)
{
	if (secondElement == nullptr && firstElement == nullptr)
		return true;

	if ((secondElement == nullptr && firstElement != nullptr) ||
		(secondElement != nullptr && firstElement == nullptr))
		return false;

	if (firstElement->name != secondElement->name)
		return false;

	bool leftPartCheck = checkElementsOnSameness(firstElement->left, secondElement-
>left);
	bool rightPartCheck = checkElementsOnSameness(firstElement->right,
secondElement->right);

	return leftPartCheck && rightPartCheck;
}



void checkTree(std::string& stringTreeForm, std::ostream& out)
{
```

```
	auto root = std::make_shared<Node>();

	//удаление пробелов
	stringTreeForm.erase(std::remove_if(stringTreeForm.begin(),
stringTreeForm.end(), [](char c) {return c == ' '; }), stringTreeForm.end());

	int checkBracketsResult = checkTreeBrackets(stringTreeForm);
	if (!checkBracketsResult)
	{
		out << "Incorrect brackets placement!\n";
		return;
	}

	if (!formTree(stringTreeForm, root))
	{
		out << "Hm, your tree is incorrect or so hard for me (i can read only
binaries trees)\n";
		return;
	}

	if (findSameElements(root->left, root) || findSameElements(root->right, root))
		out << "YES\n";
	else
		out << "NO\n";
}

int main(int argc, char** argv)
{
	if (argc > 2)
	{
		std::ifstream in(argv[1]);
		if (!in.is_open())
		{
			std::cout << "Input file is incorrect!\n";
			return 0;
		}

		std::ofstream out(argv[2]);
		if (!out.is_open()){
			std::cout << "Output file is incorrect!\n";
			return 0;
		}

		std::string stringTreeForm;

		while (std::getline(in, stringTreeForm))
		{
			if (!stringTreeForm.empty() && *(stringTreeForm.end() - 1) == '\r')
				stringTreeForm.erase(stringTreeForm.end() - 1);
			checkTree(stringTreeForm, out);
		}

		return 0;
	}

	//для консоли
	std::string stringTreeForm;

	std::cout << "Enter string: ";
	std::cin >> stringTreeForm;

	checkTree(stringTreeForm, std::cout);

	return 0;
}
```

BinaryTree.h

```
#pragma once
#include <memory>
#include <iostream>
#include <string>
#include <algorithm>
#include <fstream>

using Node = struct Node;

void printTree(std::shared_ptr<Node> const& root, std::string const&
mainRootName);
void checkTree(std::string& stringTreeForm, std::ostream& out);

//функция принимающая строку (из которой будет формироватся дерево) и ссылку на
указатель на голову дерева
bool formTree(std::string const& stringTreeForm, std::shared_ptr<Node>& root);

//функция принимающая строку и указатель на положение открывающей скобки,
//она возвращает значение содержащееся от переданной открывающей до корректной
закрывающей
std::string extractBracketsValue(std::string const& stringTreeForm, size_t*
stringIndexPointer);

//функция принимающая два узла дерева и проверяющая их на равенство(проверяется
не только значение в текущих узлах а еще и в дочерних)
bool checkElementsOnSameness(std::shared_ptr<Node> const& firstElement,
std::shared_ptr<Node> const& secondElement);

//функция проверяет наличие переданного элемента в дереве с головой root
bool treeSearch(std::shared_ptr<Node> const& checkElement, std::shared_ptr<Node>
const& root);

//функция осуществляющая перебор всех узлов дерева ищя два одинаковых при помощи
2-х вышеописанных ф-ий
bool findSameElements(std::shared_ptr<Node> const& checkElement,
std::shared_ptr<Node> const& mainRoot);

struct Node
{
  Node() = default;
  std::string name;

  std::shared_ptr<Node> left;
  std::shared_ptr<Node> right;
};
```